PriorityMessageQueue Class

Overview

The **PriorityMessageQueue** class is a thread-safe implementation of a priority queue. It allows messages to be enqueued with a priority and dequeued in the order of their priority. It uses a heap to efficiently manage the priorities of the messages. The class provides methods for enqueuing, dequeuing, peeking at the highest-priority message, checking if the queue is empty, and getting all messages in the queue.

Constructor

- __init__(self) -> None: Initializes the priority message queue.
  - heap: A list that acts as the heap to store messages with priorities.
  - lock: A threading.Lock instance for thread safety.
  - condition: A threading.Condition instance associated with the lock for synchronization purposes.

Methods

1. enqueue(self, message: Tuple[int, Any]) -> None:
   - Enqueues a message with a priority.
   - Uses heapq.heappush() to add the message to the internal heap.
   - Acquires the lock to ensure thread safety and notifies waiting threads using condition.notify().

2. dequeue(self) -> Tuple[int, Any]:
   - Dequeues and returns the message with the highest priority.
   - If the queue is empty, the method waits until a message is available using condition.wait().
   - Acquires the lock to ensure thread safety.

3. peek(self) -> Optional[Tuple[int, Any]]:
   - Returns the message with the highest priority without removing it from the queue.
   - If the queue is empty, it returns None.
   - Acquires the lock to ensure thread safety.

4. is_empty(self) -> bool:
   - Returns True if the queue is empty, otherwise returns False.
   - Acquires the lock to ensure thread safety.

5. get_all_messages(self) -> list[Tuple[int, Any]]:
   - Returns a list of all messages in the queue.
   - Acquires the lock to ensure thread safety.

ThreadPool Class

Overview

The ThreadPool class manages a pool of worker threads and allows tasks to be executed concurrently.

Constructor

- __init__(self, num_threads: int) -> None: Initializes the thread pool with the specified number of worker threads.
  - task_queue: A queue.Queue to hold tasks submitted to the thread pool.
  - threads: A list of worker threads.
  - lock: A threading.Lock instance for thread safety.

Methods

1. start(self) -> None:
   - Starts the thread pool by starting all the worker threads.

2. submit_task(self, task: Callable[[], None]) -> None:
   - Submits a task to the thread pool to be executed by one of the worker threads.
   - Acquires the lock to ensure thread safety and signals task completion using task_queue.task_done().

3. _worker(self) -> None:
   - Internal worker function that runs in each worker thread.
   - Retrieves tasks from the task queue and executes them until encountering a None task, indicating the thread should exit.

4. stop(self) -> None:
   - Stops the thread pool by adding None tasks to the task queue for each worker thread and joining all the worker threads.
   - Acquires the lock to ensure thread safety.

send_message Function

- send_message(sender: int, receiver: int, priority: int, content: Any) -> None:
   - Sends a message from one thread to another.
   - Acquires message_queue_lock to ensure thread safety.
   - Enqueues the message in the priority message queue corresponding to the receiver's thread.

Additional Functions

- simple_action(message): Performs a simple action with the given message.

- receiving_thread(thread_id): Runs in each receiving thread, continuously dequeuing messages and submitting tasks to the thread pool.

Initialization and Test

1. Initialize priority message queues for each thread.
2. Initialize a thread pool with a specified number of worker threads and start them.
3. Initialize receiving threads and start them.
4. Test the implementation by sending messages between threads.

User Input Section

- Allows the user to interactively send messages between threads.
- User can choose to continue sending messages or exit.

Optional Functionality (Commented Out)

- Peek at the highest-priority message for a specific thread.
- View the current stack of messages for a single thread.

Clean-Up

- Wait for receiving threads to finish.
- Stop the thread pool.

Note: Help of codium, github-copilot, chatgpt was taken to enhance, document and figure best actions and functionalities to program.